Developing and Describing Requirements

Petri Heiramo

Agile Coach, CST

Product Vision

- Product owner defines the overall goals in the project
- Features at very high level (”epics”)
- Defines high-level priorities
  - Features
  - Schedule, costs, quality, etc.
  - Stakeholders
- Guides all planning and project work
  - Communicate to everyone involved in the project

User Roles

- Chart out all user roles
  - Who uses the product? What are their primary interests?
  - Is there someone who will maintain it?
  - Is someone providing content or data to the system?

- Write short descriptions to capture the key elements of each user role

- Consider how you can get feedback from the various user roles
  - Direct participation, proxies, user research, ...?

- If possible, involve the user groups in the definition of the user stories

User-Driven Requirements

- Agile approaches emphasize customer-orientation
    - This slices of functionality through the system
- Therefore, also requirements should be written in a format the customer understands
    - In fact, the requirements should be *written by customers*
- User stories are considered by many as the best general approach
    - Plain language and easy to understand
    - Flexible level of detail
    - Focus on customer needs and user activities
- Implementation level detail is agreed mainly within the implementation sprints

Emerging Requirements

- Humans give best requirements as changes to existing implementation
  - People are bad at writing up-front requirement specs
    - Extra features, missing features, gold-plating, wrong priorities, ...
  - IKIWISI – ”I know it when I see it”
- Inspect and adapt
  1. Start with a simple version
  2. Ask for feedback
  3. Create an improved version
  4. Repeat 2-4 until a sufficiently good version is ready
     - The number of cycles is highly dependent on existing understanding of the subject
- **Never assume that the initial requirements are final**
  - Plan for their refinement and improvement

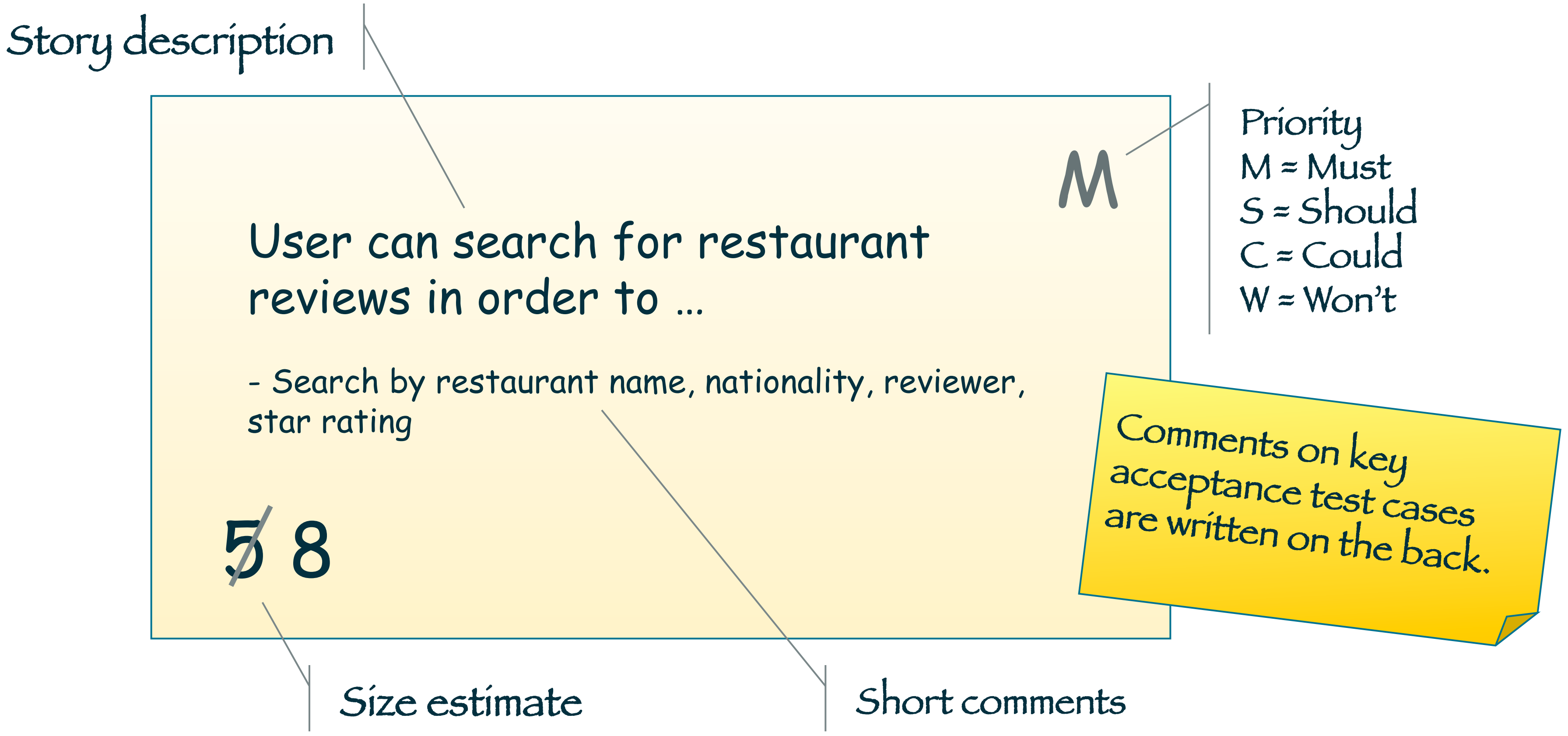
What Are User Stories?

- A concise description of desired functionality
  - More a reminder than a specification
- Three elements
  - ”Card” – the notes written on the story card
  - ”Conversation” – the discussion regarding the details of the user story
  - ”Confirmation” – notes on the key acceptance tests, e.g. Written on the backside of the story card
- Emphasize
  - Verbal communication and collaboration
  - Comprehension through the use of plain writing
  - Small size for planning
  - Deferring detail to implementation

User Stories Are NOT:

- Classic requirements, e.g.
  - 4.6) The system shall allow a company to pay for a job posting with a credit card
    - 4.6.1) The system shall accept Visa, MasterCard and American Express cards
  - These try to be absolutely accurate
    - Time-consuming, error-prone, high-detail (→ easy to lose focus of) and yield long documents
  - Imply that all features are equally valuable and necessary, at least within a priority category
- Use cases
  - Formal structure and conventions
  - Non-incremental, implied assumption for complete definition
  - User story + acceptance tests ≈ use case, assuming same scope
- Scenarios
  - Plot and narrative
  - Typically much larger than user stories
  - No tests included
- Use cases and scenarios are similar, though, and can be used as ”epics” in requirements gathering
  - They are usually split up into several user stories

Example Story Card

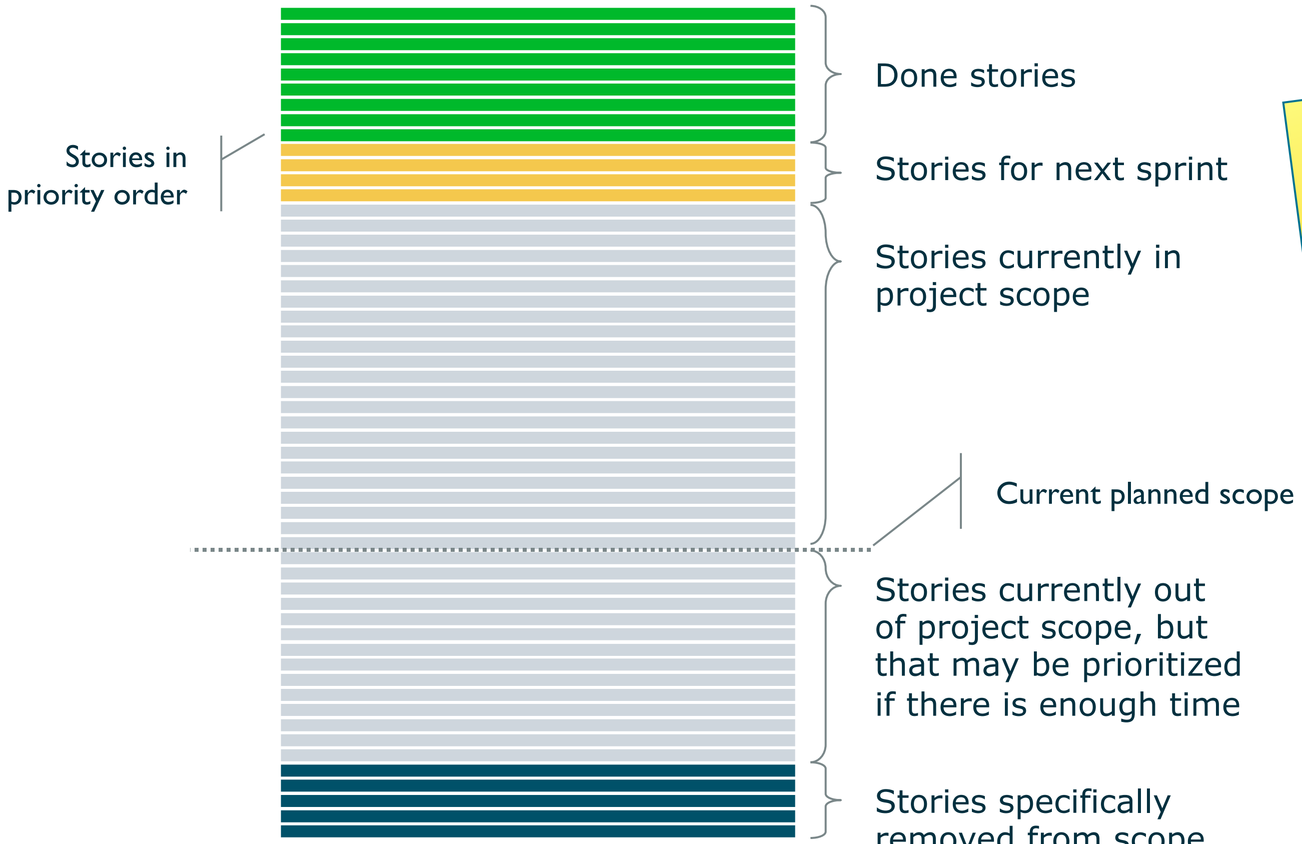


Story description
M
Priority
M = Must
S = Should
C = Could
W = Won't
User can search for restaurant reviews in order to …
- Search by restaurant name, nationality, reviewer, star rating
~~5~~ 8
Comments on key acceptance test cases are written on the back.
Size estimate
Short comments

Card? Why Not a Database?

- Most Agile projects actually use Excel or some database based tool for managing stories
  - Easy to store and filter, especially with hundreds of stories
  - Easier to use in distributed environment
  - Automatically drawn diagrams, summaries, etc.
- However, cards are superior in most planning and collaboration situations
  - Highly visual and flexible
  - Everyone can easily edit them
  - Highly effective when several people need to collaborate
- Suggestion: Use both approaches
  - Start with cards
  - Record to database at some point
  - Store the cards for additional workshops

Stories in Product Backlog


Stories in priority order
Done stories
Stories for next sprint
Stories currently in project scope
Current planned scope
Stories currently out of project scope, but that may be prioritized if there is enough time
Stories specifically removed from scope

The stories in the product backlog are ordered their relative priority, especially near the top.

Gathering User Stories

- Different approaches, depending on the source of the user stories
  - User interviews
  - Questionnaires
  - Observation
  - Story-writing workshops
- The project should probably employ user experience specialists
  - They know the techniques
- Remember feedback and continued definition throughout the project

Writing User Stories

- Standard format:

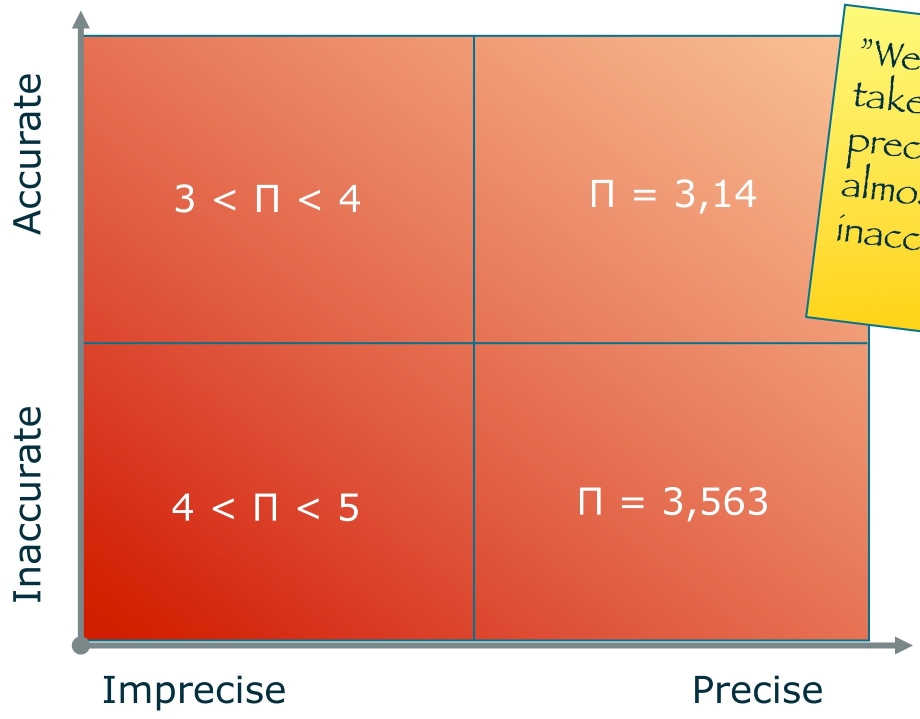
  <User> can <something> [in order to <purpose>]

  - Exceptions to the rule can be made, if the clarity of the story requires it
    - Only one user at a time must be able use one floating licence

Exercise: Estimation Quiz

- The estimated items are on a separete printout provided to you
- Please estimate ranges within which you believe the correct value is with 90% probability
  - Question: ”How tall was the tallest human ever lived?”
  - Estimate: 250-280 cm
- Do not research the answer, estimate! ☺
- You have 8 minutes, so you don’t have much time
- GO.

Accuracy vs. Precision



"We expect this project to take 374 hours" gives a very precise estimate that is almost guaranteed to be inaccurate.

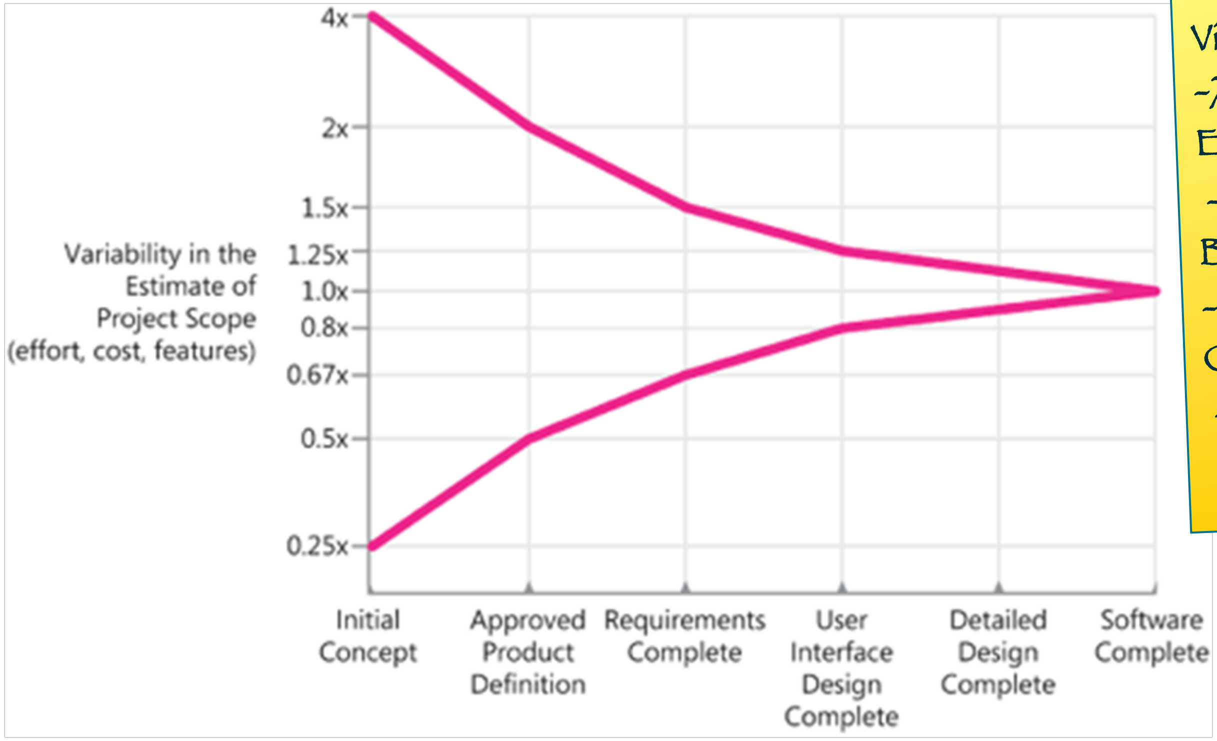
"We expect this project to take 300-400 hours" is probably accurate and gives the right impression of the precision of the estimate.

Typical Pitfalls in Estimation

- Ranges are derived mathematically, e.g. ±20%
  - Not all items have equal level of imprecision
    - You may know that the height of Mount Everest is 8850m. Giving a range for that makes no sense.
  - Not all values scale the same way
    - Numbers vs. years

- The imprecision is not indicated
  - Remember physics, i.e. the number of significant digits
    - 100 * 3.1419 = 300
  - Alternative, give ranges
    - 300 – 400 is better than 350 ± 50
    - Why?

Cone of Uncertainty



Vision estimate
-75% – +300%
Exploratory estimate
-50% – +100%
Budget estimate
-20% – +25%
Commitment estimate
-10% –+10%

© Martine Devos 2007

The Cone Doesn't Improve Itself

- Estimates improve
  - When we collect data
    - User research
    - Spikes
    - Implementation
  - Reflect on estimates
    - Remove variability
    - Making decisions
    - Keeping team stable…
- NOT by spending more time estimating
  - This tends to increase precision, not accuracy

Estimating Effort / Size

- Team estimates story size
    - Nobody else estimates size or effort

- Different options for units
    - Story points – relative units
        - Compares the size and complexity of stories against each other
        - Maintains scale through improving performance
        - Reasonable estimates can be made with very little information
        - Clearly separates size/effort and duration
    - Ideal Days – less relative, tied to time
        - Compares size against ideal performance
        - Dependent on available detail
        - Does not scale as team improves performance
        - Maybe easier to estimate for a new team

- Recommendation – Story points

Estimating Value

- For a Product Owner, it is very important to understand the business value of the stories
- Different ways to represent
  - Monetary value
  - Relative value
  - Business criticality
- Value is sometimes difficult to estimate
  - When no tool can be used, subjective estimation has to made
- Sources of value data
  - Market analyses
  - Cost analyses
  - Stakeholder interviews
  - User research

Sources of Value

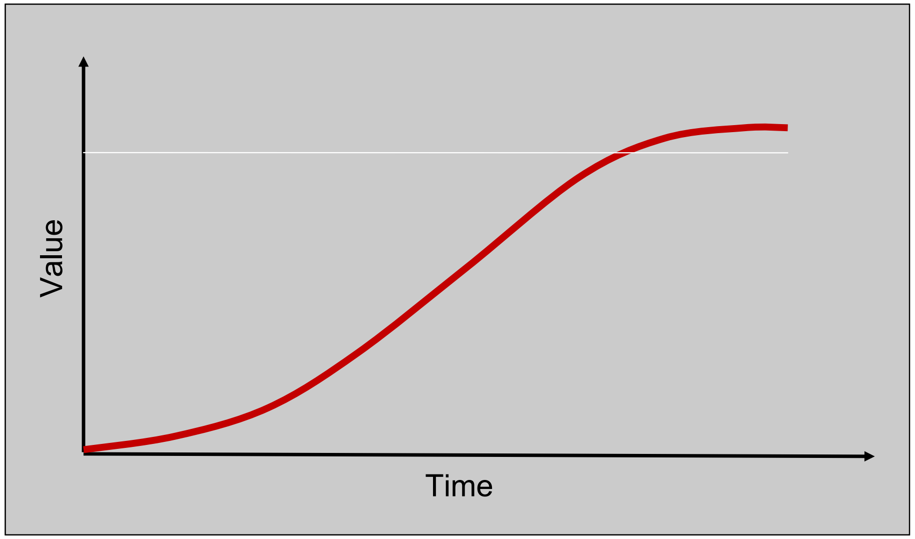
- New revenue
  - Expansion to new markets and customers
- Incremental revenue
  - Expansion within the market and existing customers
- Retained revenue
  - Avoided loss of revenue
- Operational efficiencies
  - Money saved by improving internal processes

Value Estimation Tools (examples)

- Product balance sheets
  - Compare expected income versus costs, seek highest profit
- Relative estimation tools
  - Ping-pong balls [or any other token] (”*Please distribute these 100 balls over these features, in the amounts that you value these features*”)
    - Summarizable over several respondents
    - Different amounts of balls for different user priorities
- Subjective estimation tools
  - MoSCoW (Must have, Should have, Could have, Won’t have)
  - Kano model (threshold features, linear features, exciters)
  - Feature list in priority order

Value / Cost

- To maximize value, calculate benefit ratio
- Prioritize features that return most value for time/money spent
- At some point the benefit becomes too low or even negative → end development



© Martine Devos 2007

Identifying Risk

- Four aspects of risk
  - Technical risk
    - ”We don’t yet know if it will work or what it will cost.”
  - Architectural risk
    - ”We are unsure if we can meet the performance targets in the server.”
    - ”This will be difficult to integrate to the rest of the system.”
  - Usability risk
    - ”Can we make this usable enough for people to use it effectively?”
  - Business risk
    - ”I don’t know if my customers will like this.”
    - ”If we don’t get this out on time, our competition may get the market.”
- If necessary, also traditional risk identification processes can be used
- Traditional risk valuation tools and scales can also be used in Scrum projects

Mitigating / Eliminating Risk

- The Agile way to reduce risk is to implement
  - We learn either way
  - We can get feedback

- Prioritize high risk items early in the project
  - If the risks realize, you know it early
    - Fail early, fail cheap
  - If you succeed, you can go forward with confidence

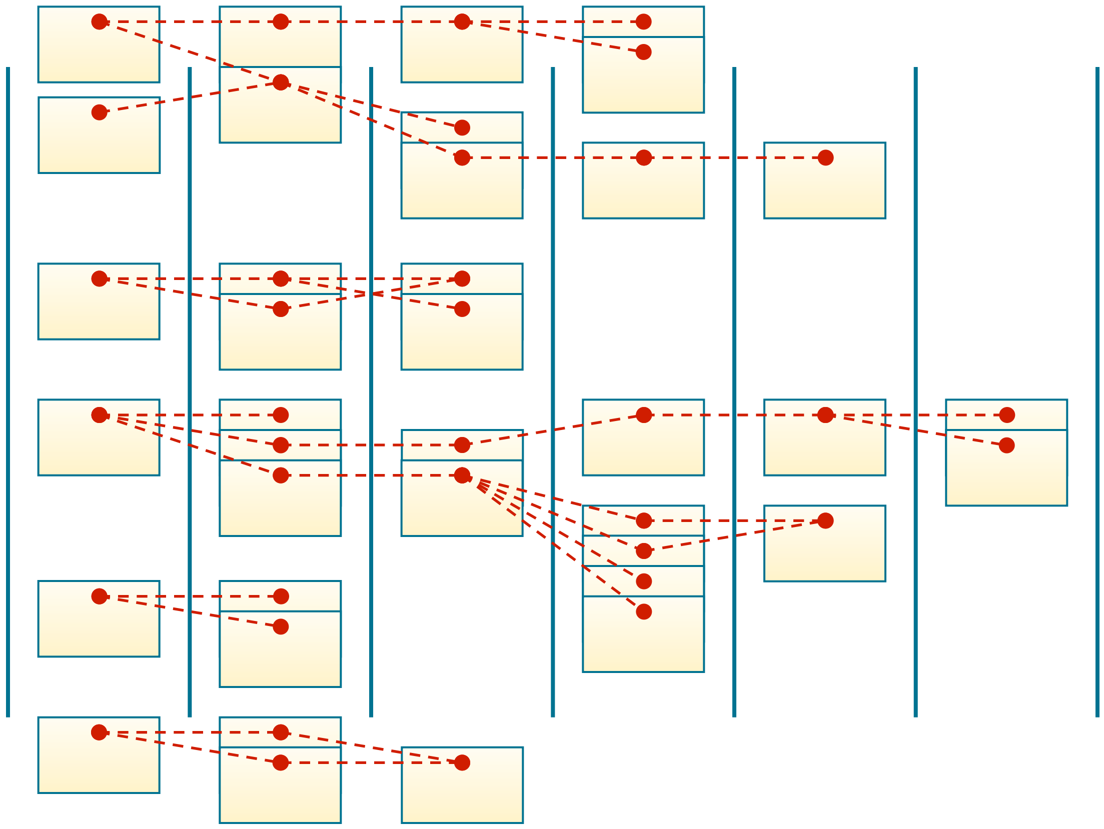
Elimination by Risk Type

- Technical risk
  - Implement simple versions of the features
- Architectural risk
  - Select stories that implement architectural elements all around the envisioned architecture
  - Select stories that require the team to prove e.g. system performance
- Business risk
  - Implement an initial version of the risky feature
  - Ask feedback from stakeholders/users, or conduct usability or user studies

Dependencies

- It is often useful to identify dependencies between the stories
    - ”Critical path”
    - Where to start
    - Fully independent stories
- In small projects, much of this is quite obvious or simple at least
- Large projects may have to use heavier tools which support the management of the dependencies in digital form
- One collaborative way to identify dependencies is to use a dependency map

Example of a Dependency Map



Spikes

- Spike = a short study to clarify something
- Spikes are very useful for
  - Testing something new
  - Breaking up large stories
  - Studying alternatives
  - Estimating difficult-to-estimate stories
- Avoid analysis paralysis
  - Spike should as small as it can be & time-boxed
  - Avoid sprints with many spikes
    - Always balance study with practical implementation
  - Spikes should always translate into action
    - Stories split, stories estimated, decisions made

Thank you

For more info:

petri.heiramo@gmail.com